ST. XAVIER’S COLLEGE

**(Affiliated to Tribhuvan University)**

**Maitighar, Kathmandu**

****

**Database Management System**

**Theory Assignment**

**SUBMITTED BY:**

Sneha Prasai

013BSCCSIT040

**SUBMITTED TO**

|  |  |
| --- | --- |
| **Er. Sanjay Kr. Yadav**  **( Lecturer )** |  |
| **Department of Computer Science** | |

1. **GRANT and REVOKE authorization:**

### The grant statement is used to confer authorization

### grant<privilege list>

### on<relation name or view name>to <user list>

### <user list> is:

### auser­id

### public, which allows all valid users the privilege granted

### A role

### Granting a privilege on a view does not imply granting any privilegeson the underlying relations.The grantor of the privilege must already hold the privilege on the specified item (or be the database administrator).

### Privileges:

### select: allows read access to relation,or the ability to query using the view

### Example: grant users U1, U2, and U3 select authorization on the branch relation:

### grant select on branch to U1, U2, U3

### insert: the ability to insert tuples

### update: the ability to update using the SQL update statement

### delete: the ability to delete tuples.

### all privileges: used as a short form for all the allowable privileges

### REVOKE authorization

### The revoke statement is used to revoke authorization.

### revoke<privilege list>

### on<relation name or view name>from <user list>

### Example:

### revoke select on branch from U1, U2, U3

### <privilege­list> may be all to revoke all privileges the revoke may hold.

### If <revokee­list> includes public, all users lose the privilege except those granted it explicitly.

### If the same privilege was granted twice to the same user by different grantees, the user may retain the privilege after the revocation.

### All privileges that depend on the privilege being revoked are also revoked.

1. **Data encryption**

Encrypting sensitive data in databases has clearly gone beyond optional, and is now a firm requirement. Whether an organization is looking to secure intellectual property, comply with privacy or regulatory mandates, or simply guard the organization’s brand against the damage associated with data breaches, database encryption represents a vital imperative.

By providing database encryption for sensitive data in databases, organizations can establish a strong line of defense that can help secure sensitive assets against a range of threats. However, while the reasons to adopt database encryption are clear, that doesn’t mean the effort is simple. In fact, for many organizations, database encryption has presented a range of obstacles, including degraded database performance, laborious revisions to application code, and complex and time consuming key management efforts.

## Vormetric Transparent Encryption

## Vormetric Application Encryption

## Key Management for Oracle and Microsoft SQL Server Database Encryption

Data Encryption helps to save data from following attacks:

* **Virtual attack**
* **Physical attack**
* **Power**
* **Flexibility**
* **Transparency**

1. **Transivity, Reflexivity, and Augmentation properties of FDs**

Given that *X*, *Y*, and *Z* are sets of attributes in a relation *R*, one can derive several properties of functional dependencies. Among the most important are the following, usually called [Armstrong's axioms](https://en.wikipedia.org/wiki/Armstrong%27s_axioms):

* **Reflexivity**: If *Y* is a subset of *X*, then *X* → *Y*
* **Augmentation**: If *X* → *Y*, then *XZ* → *YZ*
* **Transitivity**: If *X* → *Y* and *Y* → *Z*, then *X* → *Z*

"Reflexivity" can be weakened to just X \rightarrow \varnothing, i.e. it is an actual [axiom](https://en.wikipedia.org/wiki/Axiom), where the other two are proper [inference rules](https://en.wikipedia.org/wiki/Inference_rules), more precisely giving rise to the following rules of syntactic consequence:

\vdash X \rightarrow \varnothing  
X \rightarrow Y \vdash XZ \rightarrow YZ  
X \rightarrow Y, Y \rightarrow Z \vdash X \rightarrow Z.

These three rules are a [sound](https://en.wikipedia.org/wiki/Soundness) and [complete](https://en.wikipedia.org/wiki/Completeness_(logic)) axiomatization of functional dependencies. This axiomatization is sometimes described as finite because the number of inference rules is finite, with the caveat that the axiom and rules of inference are all [schemata](https://en.wikipedia.org/wiki/Schema_(logic)), meaning that the *X*, *Y* and *Z* range over all ground terms (attribute sets).

1. **BCNF and decomposition into BCNF**

BCNF:

We say a relation R is in BCNF if whenever X → Y is a nontrivial FD that holds in R, X is a superkey. Remember: nontrivial means Y is not contained in X. Remember, a superkey is any superset of a key (not necessarily a proper superset)

Example:

Drinkers(name, addr, beersLiked, manf, favBeer)

FD’s: name → addr favBeer, beersLiked → manf

Only key is {name, beersLiked}

In each FD, the left side is not a superkey

Any one of these FD’s shows Drinkers is not in BCNF

Decomposition into BCNF:

Given: relation R with FD’s F § Look among the given FD’s for a BCNF violation X → Y

If any FD following from F violates BCNF, then there will surely be an FD in F itself that violates BCNF

Compute X+

Not all attributes, or else X is a superkey.

1. **Characteristics schedules based on Recoverability**

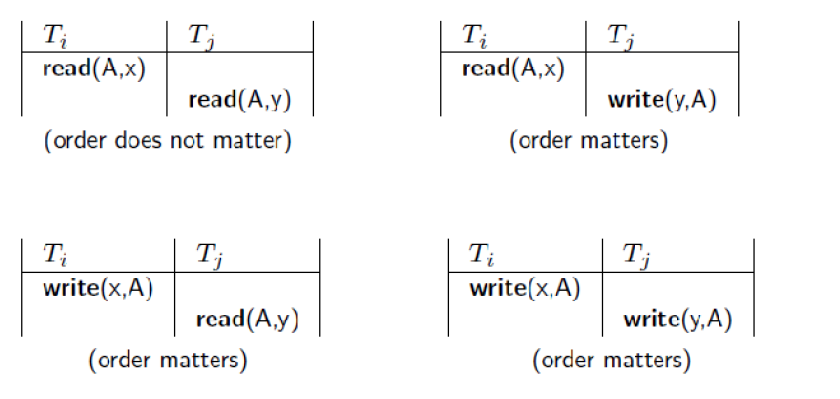
When transactions are executing concurrently in an interleaved fashion, the order of execution of  
operations from the various transactions forms what is known as a transaction schedule (or  
history).  
A schedule (or history) S of n transactions T1, T2, …, Tn:  
It is an ordering of the operations of the transactions subject to the constraint that, for each  
transaction Ti that participates in S, the operations of T1 in S must appear in the same order in  
which they occur in T1.  
Note, however, that operations from other transactions Tj can be interleaved with the operations  
of Ti in S.

1. **Characteristics schedules based on Serializabiliy:**

DBMS must control concurrent execution of transactions to ensure read consistency, i.e., to avoid dirty reads etc. A (possibly concurrent) schedule S is serializable if it is equivalent to a serial schedule S0, i.e., S has the same result database state as S0.

How to ensure serializability of concurrent transactions?

Conflicts between operations of two transactions:



A schedule S is serializable with regard to the above conflicts iff S can be transformed into a serial schedule S' by a series of swaps of non-conflicting operations. Checks for serializability are based on precedence graph that describes dependencies among  
concurrent transactions; if the graph has no cycle, and then the transactions are serializable. - they can be executed concurrently without affecting each other’s transaction result.

1. **Transactions supports in SQL**

A **single** SQL statement is always considered to be **atomic**.

* + Either the statement completes execution without error or it fails and leaves the database unchanged.

With SQL, there is no explicit Begin Transaction statement.

* + Transaction initiation is done implicitly when particular SQL statements are encountered.

Every transaction must have an explicit end statement, which is either a COMMIT or ROLLBACK.

Characteristics specified by a SET TRANSACTION statement in SQL2:

**Access mode**:

READ ONLY or READ WRITE.

* + - The default is READ WRITE unless the isolation level of READ UNCOMITTED is specified, in which case READ ONLY is assumed.

**Diagnostic size** n, specifies an integer value n, indicating the number of conditions that can be held simultaneously in the diagnostic area.

Characteristics specified by a SET TRANSACTION statement in SQL2 (contd.):

**Isolation level** <isolation>, where <isolation> can be READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ or SERIALIZABLE. The default is SERIALIZABLE.

* + With SERIALIZABLE: the interleaved execution of transactions will adhere to our notion of serializability.
  + However, if any transaction executes at a lower level, then serializability may be violated.

Potential problem with lower isolation levels:

**Dirty Read**:

* + Reading a value that was written by a transaction which failed.

**Nonrepeatable Read**:

* + Allowing another transaction to write a new value between multiple reads of one transaction.
  + A transaction T1 may read a given value from a table. If another transaction T2 later updates that value and T1 reads that value again, T1 will see a different value.

Consider that T1 reads the employee salary for Smith. Next, T2 updates the salary for Smith. If T1 reads Smith's salary again, then it will see a different value for Smith's salary.

Phantoms:

New rows being read using the same read with a condition.

* + - * A transaction T1 may read a set of rows from a table, perhaps based on some condition specified in the SQL WHERE clause.
      * Now suppose that a transaction T2 inserts a new row that also satisfies the WHERE clause condition of T1, into the table used by T1.
      * If T1 is repeated, then T1 will see a row that previously did not exist, called a phantom.